iani

IOANNIS ZANNOS

March 5, 2015

Contents

Generic Emacs Customization

Theme (experimental)

```
(require 'moe-theme)
(moe-dark)
```

Font (experimental)

```
(add-to-list 'default-frame-alist
             '(font . "Anonymous Pro-14"))

(defun larger-frame-font ()
  (interactive)
  (set-face-attribute
   'default nil
   :height
   (+ (face-attribute 'default :height) 10)) )

(defun smaller-frame-font ()
  (interactive)
  (set-face-attribute
   'default nil
   :height
   (- (face-attribute 'default :height) 10)) )

(global-set-key (kbd "C-c C--") 'smaller-frame-font)

(global-set-key (kbd "C-c C-+") 'larger-frame-font)
```

Prelude stuff

Turn off guru mode See: http://stackoverflow.com/questions/13263717/unable-to-set-off-the-guru-mode-in-emacs-prelude

```
(guru-mode -1)
(guru-global-mode -1)
(setq prelude-guru nil)
(add-hook 'prelude-prog-mode-hook
          (lambda ()
            (guru-mode -1)) t)
```

Prelude-copy-filename-to-clipboard

```
(global-set-key (kbd "C-c p C-c") 'prelude-copy-file-name-to-clipboard)
```

Use visible bell instead of beep

```
(setq visible-bell t)
```

Blink cursor

```
(blink-cursor-mode 1)
```

Internationalization: Set Mac OS native fonts for japanese/greek

Here is a solution that works fine for displaying Latin, Greek and Japanese texts in the same buffer. The Menlo font displays Greek clearly and smoothly, and since it is a mono-space font, it also works well in tables. The two lisp expressions have to be evaluated in the following order for this to work.

Note: This is necessary if the latin font is set to some font that does not have a full greek character set.

```
(set-fontset-font "fontset-default"
                  'japanese-jisx0208
                  '("Hiragino Mincho Pro" . "iso10646-1"))
(set-fontset-font "fontset-default"
                  'greek
 ;; Note: iso10646-1 = Universal Character set (UCS)
 ;; It is compatible to Unicode, in its basic range
                  '("Menlo" . "iso10646-1"))
```

Links for further reading:
http://sunsite.univie.ac.at/textbooks/emacs/emacs_22.html#SEC203 http://www.gnu.org/software/emacs/manual/html_node/emacs/Modifying-Fontsets.html For Greek see: http://iris.math.aegean.gr/~atsol/emacs-unicode/

Maximize/toggle frame

```
;; (maximize-frame) ;; maximize frame on startup
(defun toggle-fullscreen ()
  "Toggle full screen"
  (interactive)
  (set-frame-parameter
   nil 'fullscreen
   (when (not (frame-parameter nil 'fullscreen)) 'fullboth)))

(tool-bar-mode -1)

(global-set-key (kbd "H-f") 'toggle-fullscreen)

(toggle-fullscreen)
```

```
(require 'maxframe) ;; (maximize-frame) command/function
```

Toggle visual line mode

Useful for org-mode todo agenda.

```
(global-set-key (kbd "H-h v") 'visual-line-mode)
```

Movement: backward-sentence, forward-sentence

Bind backward-sentence and forward-sentence in 2 different ways:

1. Control-shift-f and b in analogy to control-b/f and meta-b/f (character and word backward and forward).
2. Meta-[and] in analogy to Meta-shift-[and] (org-element backward and forward).

```
(global-set-key (kbd "M-B") 'backward-sentence)
(global-set-key (kbd "M-F") 'forward-sentence)
(global-set-key (kbd "M-[") 'backward-sentence)
(global-set-key (kbd "M-]") 'forward-sentence)
```

Insert timestamp (C-c C-x t)

```
(defun insert-timestamp (&optional short-type)
  "Insert a timestamp."
  (interactive "P")
  (if short-type
      (insert
       (let ((date (calendar-current-date)))
         (format "%s. %s. %s"
                 (nth 1 date)
                 (nth 0 date)
                 (nth 2 date))))
   (insert (format-time-string "%a, %b %e %Y, %R %Z"))))

(global-set-key (kbd "C-c C-x t") 'insert-timestamp)
```

Generic Packages

Require org-drill

```
(require 'org-drill)
```

Install el-get for installing of packages from github etc.
See: `https://github.com/dimitri/el-get/blob/master/README.md`

```
(add-to-list 'load-path "~/.emacs.d/el-get/el-get")

(unless (require 'el-get nil 'noerror)
  (with-current-buffer
      (url-retrieve-synchronously
       "https://raw.github.com/dimitri/el-get/master/el-get-install.el")
    (goto-char (point-max))
    (eval-print-last-sexp)))

(add-to-list 'el-get-recipe-path "~/.emacs.d/el-get-user/recipes")
(el-get 'sync)
```

Bring elisp up-to-date: dash.el

A modern list api for Emacs. No 'cl required.

See: `https://github.com/magnars/dash.el#functions`
Used in my packages as well as in projectile and other packages.
I load it here to have it available when experimenting with code.

```
(require 'dash)
```

desktop
Save desktop between sessions. To clear desktop: M-x desktop-clear.
Note: As of Mon, Dec 1 2014, 11:19 EET this creates error with sentinel process on startup. Disabled.

```
(desktop-save-mode 1)
```

breadcrumb, bookmark+
See: `http://breadcrumbemacs.sourceforge.net/news.html`

```
(require 'breadcrumb)

;; (global-set-key [(shift space)]         'bc-set)              ;; Shift-SPACE for set bookmark
(global-set-key (kbd "S-SPC")            'bc-set) ;; Shift-SPACE for set bookmark
(global-set-key [(meta j)]              'bc-previous)       ;; M-j for jump to previous
(global-set-key [(shift meta j)]        'bc-next)           ;; Shift-M-j for jump to next
(global-set-key [(meta up)]             'bc-local-previous) ;; M-up-arrow for local previous
(global-set-key [(meta down)]           'bc-local-next)     ;; M-down-arrow for local next
(global-set-key [(control c)(j)]        'bc-goto-current)   ;; C-c j for jump to current bookmark
(global-set-key [(control x)(meta j)]   'bc-list)           ;; C-x M-j for the bookmark menu list

(require 'desktop)
(require 'bookmark+)

(setq bookmark-default-file
```

```
      "~/.emacs.d/personal/bookmarks/default-bookmarks.bmk")

(defun bookmark-save-named (&optional name)
  "mod of bookmark-save to save bookmark under name
in one default directory in users prelude folder."
  (interactive "Mbookmark filename: ~/.emacs.d/personal/bookmarks/: ")
  (let ((path
         (file-truename
          (concat
           "~/.emacs.d/personal/bookmarks/"
           (replace-regexp-in-string "/" "_" name)
           ".bmk"))))
    (setq bmkp-current-bookmark-file path)
    (bookmark-save)))

(global-set-key (kbd "C-x r C-s") 'bookmark-save-named)

(defun bmkp-desktop-save-named (&optional name)
  "mod of bmkp-desktop-save to save desktop bookmark under name
in under one default directory in users prelude folder."
  (interactive "MSave desktop ~/.emacs/personal/bookmarks/desktops/?: ")
  (let ((path
         (file-truename
          (concat
           "~/.emacs.d/personal/bookmarks/desktops/"
           (replace-regexp-in-string "/" "_" name)
           ".desktop"))))
    (bmkp-desktop-save path)
    (let ((bookmark-make-record-function
           (lexical-let ((df path))
             (lambda () (bmkp-make-desktop-record df))))
          (current-prefix-arg 99)) ; Use all bookmarks for completion, for `bookmark-set'.
      (bookmark-set name))))

(defun bmkp-load-auto-saved-desktop ()
  (interactive)
;;  (bookmark-bmenu-list) ;; needed to update list if never loaded
  (bmkp-desktop-jump "auto-save-desktop"))

(add-hook 'kill-emacs-hook
          (lambda () (bmkp-desktop-save-named "auto-save-desktop")))

(global-set-key (kbd "C-x r C-k") 'bmkp-desktop-save-named)
(global-set-key (kbd "C-x p r") 'bookmark-rename)
```

```
(define-key bookmark-bmenu-mode-map "r" 'bookmark-rename)
(global-set-key (kbd "C-x j M-k") 'bmkp-load-auto-saved-desktop)

(bookmark-bmenu-list) ;; make sure bookmark list is loaded
```

Completion help: icicles, imenu+, auto-complete, ido, guide-key

```
(require 'ido)
(require 'flx-ido)
(require 'imenu+)
(require 'auto-complete)
(ido-mode t)
(ido-vertical-mode t)
(icicle-mode) ;; breaks dired? Tue, Nov  4 2014, 19:17 EET
;; guide-key causes erratic delays when posting in ths SC post buffer
;; from sclang.  Therefore disabled.
;; (require 'guide-key)
;; (setq guide-key/guide-key-sequence '("C-x r" "C-x 4" "H-h" "H-m" "H-p" "H-d" "C-c"))
;;  (guide-key-mode 1)  ; Enable guide-key-mode
;; (yas-global-mode) ; interferes with auto-complete in elisp mode.
```

Buffer-move, windmove, buffer switching

Use cursor keys to switch cursor position between windows. Bound to `control-super-<cursorkey>`.

Bound to `function-super-<cursor key>`

windmove

```
(require 'windmove)
(global-set-key (kbd "H-{") 'windmove-up)
(global-set-key (kbd "H-}") 'windmove-down)
(global-set-key (kbd "H-]") 'windmove-right)
(global-set-key (kbd "H-[") 'windmove-left)

(require 'buffer-move)
(global-set-key (kbd "<S-prior>") 'buf-move-up)
(global-set-key (kbd "<S-next>") 'buf-move-down)
(global-set-key (kbd "<S-end>") 'buf-move-right)
(global-set-key (kbd "<S-home>") 'buf-move-left)

(global-set-key (kbd "<s-home>") 'previous-buffer)
(global-set-key (kbd "<s-end>") 'next-buffer)
```

Completion help: icicles, imenu+, auto-complete, ido, guide-key

```
(require 'ido)
(require 'flx-ido)
```

```
(require 'imenu+)
(require 'auto-complete)
(ido-mode t)
(ido-vertical-mode t)
(icicle-mode)
;; guide-key causes erratic delays when posting in ths SC post buffer
;; from sclang.  Therefore disabled.
;; (require 'guide-key)
;; (setq guide-key/guide-key-sequence '("C-x r" "C-x 4" "H-h" "H-m" "H-p" "H-d" "C-c"))
;;  (guide-key-mode 1)  ; Enable guide-key-mode
;; (yas-global-mode) ; interferes with auto-complete in elisp mode.
```

File-system navigation: projectile, helm

1. projectile

```
(setq projectile-completion-system 'grizzl)
(setq *grizzl-read-max-results* 40)
(defun projectile-dired-project-root ()
  "Dired root of current project.  Can be set as value of
projectile-switch-project-action to dired root of project when switching.
Note: projectile-find-dir (with grizzl) does not do this, but it
asks to select a *subdir* of selected project to dired."
  (interactive)
  (dired (projectile-project-root)))

(setq projectile-switch-project-action 'projectile-commander)

(defun projectile-post-project ()
  "Which project am I actually in?"
  (interactive)
  (message (projectile-project-root)))

(defun projectile-add-project ()
  "Add folder of current buffer's file to list of projectile projects"
  (interactive)
  (if (buffer-file-name (current-buffer))
      (projectile-add-known-project
       (file-name-directory (buffer-file-name (current-buffer))))))

(global-set-key (kbd "H-p c") 'projectile-commander)
(global-set-key (kbd "H-p h") 'helm-projectile)
(global-set-key (kbd "H-p s") 'projectile-switch-project)
(global-set-key (kbd "H-p d") 'projectile-find-dir)
(global-set-key (kbd "H-p f") 'projectile-find-file)
```

```
(global-set-key (kbd "H-p w") 'projectile-post-project)
(global-set-key (kbd "H-p D") 'projectile-dired-project-root)
(global-set-key (kbd "H-p +") 'projectile-add-project)
(global-set-key (kbd "H-p -") 'projectile-remove-known-project)
(global-set-key (kbd "H-p a") 'projectile-ack) ;; better search than grep
```

2. helm

   NOTE: helm-swoop must be installed from: `https://raw.github.com/ShingoFukuyama/helm-swoop/master/helm-swoop.el` or `https://raw.github.com/ShingoFukuyama/helm-swoop/`

```
    ;; must call these to initialize  helm-source-find-files

    (require 'helm-files) ;; (not auto-loaded by system!)
;;    (require 'helm-projectile)
    (require 'helm-swoop) ;; must be put into packages
    ;; Don't bicker if not in a project:
    (setq projectile-require-project-root)

    ;; Added by IZ following this:
    ;; https://github.com/emacs-helm/helm/issues/604
    ;; :

    (add-hook 'helm-find-files-before-init-hook
              (lambda ()
                (progn
                  ;; List Hg files in project.
                  (helm-add-action-to-source-if
                   "Hg list files"
                   'helm-ff-hg-find-files
                   helm-source-find-files
                   'helm-hg-root-p)
                  ;; Byte compile files async
                  (helm-add-action-to-source-if
                   "Byte compile file(s) async"
                   'async-byte-compile-file
                   helm-source-find-files
                   'helm-ff-candidates-lisp-p)
                  ;; Add add-to-projectile action after helm-find-files.
                  (let ((find-files-action (assoc 'action helm-source-find-files)))
                    (setcdr find-files-action
                            (cons
                             (cadr find-files-action)
                             (cons '("Add to projectile" . helm-add-to-projectile)
```

```
                                    (cddr find-files-action))))))))

   ;; Use helm-find-files actions in helm-projectile
;;   (let ((projectile-files-action (assoc 'action helm-source-projectile-files-list)))
;;       (setcdr projectile-files-action (cdr (assoc 'action helm-source-find-files))))

   (defun helm-add-to-projectile (path)
     "Add directory of file to projectile projects.
   Used as helm action in helm-source-find-files"
     (projectile-add-known-project (file-name-directory path)))

   (global-set-key (kbd "H-h p") 'helm-projectile)
   (global-set-key (kbd "H-h g") 'helm-do-grep)
   (global-set-key (kbd "H-h f") 'helm-find-files)
   (global-set-key (kbd "H-h r") 'helm-resume)
   (global-set-key (kbd "H-h b") 'helm-bookmarks)
   (global-set-key (kbd "H-h l") 'helm-buffers-list)
   (global-set-key (kbd "H-M-h") 'helm-M-x)
   (global-set-key (kbd "H-h w") 'helm-world-time)
   (global-set-key (kbd "H-h s") 'helm-swoop)
   (global-set-key (kbd "C-c m") 'helm-mini)

   (setq display-time-world-list
         '(("America/Los_Angeles" "Santa Barbara")
           ("America/New_York" "New York")
           ("Europe/London" "London")
           ("Europe/Lisbon" "Lisboa")
           ("Europe/Madrid" "Barcelona")
           ("Europe/Paris" "Paris")
           ("Europe/Berlin" "Berlin")
           ("Europe/Rome" "Rome")
           ;; ("Europe/Albania" "Gjirokastra") ;; what city to name here?
           ("Europe/Athens" "Athens")
           ("Asia/Calcutta" "Kolkatta")
           ("Asia/Jakarta" "Jakarta")
           ("Asia/Shanghai" "Shanghai")
           ("Asia/Tokyo" "Tokyo")))
```

Note on icicle key bindings and org-mode

C-c ' in org mode runs the command org-edit-special, for editing babel commands and other blocks. To avoid conflict with icicles binding of the same key to icicle-occur, remap the latter to something else (e.g. C-c C-M-'), like this:

1. type M-x customize-group <RET> Icicles-Key-Bindings <RET>

2. Scroll down to Icicle Top Level Key Bindings, open the list, find icicle-occur, enter C-c C-M-' to the `Key:` field, go to top of buffer, use the `State` button to save this.

See also discussion here: `http://www.emacswiki.org/emacs/Icicles_-_Key_Binding_Discussion`

lacarte: select menu items from the keyboard (good for org-mode with imenu)

```
(require 'lacarte)
;; (global-set-key [?\e ?\M-x] 'lacarte-execute-command)
```

Ido-imenu command and jump back after completion, by Magnar Sveen, and others.

Disabled.

```
;;; ido-imenu
(defun ido-imenu ()
  "Update the imenu index and then use ido to select a symbol to navigate to.
Symbols matching the text at point are put first in the completion list."
  (interactive)
  (imenu--make-index-alist)
  (let ((name-and-pos '())
        (symbol-names '()))
    (flet ((addsymbols
          (symbol-liost)
          (when (listp symbol-list)
            (dolist (symbol symbol-list)
              (let ((name nil) (position nil))
                (cond
                 ((and (listp symbol) (imenu--subalist-p symbol))
                  (addsymbols symbol))

                 ((listp symbol)
                  (setq name (car symbol))
                  (setq position (cdr symbol)))

                 ((stringp symbol)
                  (setq name symbol)
                  (setq position
                        (get-text-property 1 'org-imenu-marker symbol))))

                (unless (or (null position) (null name))
                  (add-to-list 'symbol-names name)
                  (add-to-list 'name-and-pos (cons name position))))))))
      (addsymbols imenu--index-alist))
```

```
;; If there are matching symbols at point, put them at the beginning of `symbol-names'.
    (let ((symbol-at-point (thing-at-point 'symbol)))
      (when symbol-at-point
        (let* ((regexp (concat (regexp-quote symbol-at-point) "$"))
               (matching-symbols
                (delq nil (mapcar (lambda (symbol)
                                    (if (string-match regexp symbol) symbol))
                                  symbol-names))))
          (when matching-symbols
            (sort matching-symbols (lambda (a b) (> (length a) (length b))))
            (mapc
             (lambda (symbol)
               (setq symbol-names (cons symbol (delete symbol symbol-names))))
             matching-symbols)))))
    (let* ((selected-symbol (ido-completing-read "Symbol? " symbol-names))
           (position (cdr (assoc selected-symbol name-and-pos))))
      (goto-char position))))

;; Push mark when using ido-imenu

(defvar push-mark-before-goto-char nil)

(defadvice goto-char (before push-mark-first activate)
  (when push-mark-before-goto-char
    (push-mark)))

(defun ido-imenu-push-mark ()
  (interactive)
  (let ((push-mark-before-goto-char t))
    (ido-imenu)))
```

smex (auto-complete minibuffer commands called with Meta-x)

Note: since March 2014 I mostly use helm-M-x (bound to Hyper-meta-x) instead of Meta-x, so smex is not crucial.

```
;; Smex: Autocomplete meta-x command
(global-set-key [(meta x)]
                (lambda ()
                  (interactive)
                  (or (boundp 'smex-cache)
                      (smex-initialize))
                  (global-set-key [(meta x)] 'smex)
                  (smex)))

(global-set-key [(shift meta x)]
```

```
                (lambda ()
                  (interactive)
                  (or (boundp 'smex-cache)
                      (smex-initialize))
                  (global-set-key [(shift meta x)] 'smex-major-mode-commands)
                  (smex-major-mode-commands)))
```

Multiple Cursors

```
(require 'multiple-cursors)
(global-set-key (kbd "C-S-c C-S-c") 'mc/edit-lines)
(global-set-key (kbd "C->") 'mc/mark-next-like-this)
(global-set-key (kbd "C-<") 'mc/mark-previous-like-this)
(global-set-key (kbd "C-M->") 'mc/mark-more-like-this-extended)
(global-set-key (kbd "C-c C-<") 'mc/mark-all-like-this)
;; (global-set-key (kbd "C->") 'mc/mark-next-symbol-like-this)
;; (global-set-key (kbd "C->") 'mc/mark-next-word-like-this)
```

Whitespace Mode

```
(defun turn-off-whitespace-mode () (whitespace-mode -1))
(defun turn-on-whitespace-mode () (whitespace-mode 1))
```

Key Chords

```
(require 'key-chord)
(key-chord-mode 1)

(defun paren-sexp ()
  (interactive)
  (insert "(")
  (forward-sexp)
  (insert ")"))

(defun code-quote-sexp ()
  (interactive)
  (insert "=")
  (forward-sexp)
  (insert "="))

(key-chord-define-global "jk"     'ace-jump-char-mode)
(key-chord-define-global "jj"     'ace-jump-word-mode)
(key-chord-define-global "jl"     'ace-jump-line-mode)

(key-chord-define-global "hj"     'undo)
```

```
(key-chord-define-global "{}"     "{   }\C-b\C-b\C-b")
(key-chord-define-global "()"     'paren-sexp)
(key-chord-define-global "(_"     "()\C-b")
(key-chord-define-global "-="     'code-quote-sexp)
;; to add: quote, single quote around word/sexp
;; Exit auto-complete, keeping the current selection,
;; while avoiding possible side-effects of TAB or RETURN.
(key-chord-define-global "KK"      "\C-f\C-b")
;; Trick for triggering yasnippet when using in tandem with auto-complete:
;; Move forward once to get out of auto-complete, then backward once to
;; end of keyword, and enter tab to trigger yasnippet.
(key-chord-define-global "KL"      "\C-f\C-b\C-i")

;; Jump to any symbol in buffer using ido-imenu
(key-chord-define-global "KJ"      'ido-imenu)
```

hl-sexp mode (also: highlight-sexps)
Highlight expressions enclosed by (), {} or [] in code.
There exist 2 versions:

1. hl-sexp package available from elpa. Package name: hl-sexp Mode name: hl-sexp-mode
2. highlight-sexps.el, from `http://www.emacswiki.org/emacs/HighlightSexp`. Package name: highlight-sexps Mode name: highlight-sexps-mode

highlight-sexps.el looks nicer, because it highlights both the innermost s-expression and the one enclosing it, and it does not un-highlight the line where the cursor is on. But it sometimes stops working. So I use hl-sexp

```
(require 'hl-sexp)
;; (require 'highlight-sexps)
;; Include color customization for dark color theme here.
(custom-set-variables
 '(hl-sexp-background-colors (quote ("gray0"  "#0f003f"))))
```

Directory/Buffer navigation: Dired+, Dirtree, Speedbar

1. Dired+, Dirtree, Speedbar

   Note about dirtree: Very handy. There are several versions out there, and there is also a different package under the same name. Not all versions work. This one works for me: `https://github.com/rtircher/dirtree`. I installed it manually (not via `el-get`, el-get's registered versions of dirtree resulted in conflicts. Dirtree is similar to file-browse mode of speedbar, but it servers a different purpose: With dirtree you can select one or more directories to browse, and keep them all in the sidebar. Speedbar always shows only the directory of the file of the current buffer.

```
;;  (require 'dired+)
  (require 'dirtree)
  (global-set-key (kbd "H-d d") 'dirtree-show)
  ;; sr-speedbar is broken in emacs 24.4.1
  ;; (require 'sr-speedbar)
  ;; (speedbar-add-supported-extension ".sc")
  ;; (speedbar-add-supported-extension ".scd")
  ;; (global-set-key (kbd "H-d H-s") 'sr-speedbar-toggle)
```

2. Open pdf files with default macos app in dired

   From: http://stackoverflow.com/questions/20019732/define-keybinding-for-dired-to-run-a-command-open-o

```
(define-key dired-mode-map (kbd "<SPC>")
  (lambda () (interactive)
    (let ((lawlist-filename (dired-get-file-for-visit)))
      (if (equal (file-name-extension lawlist-filename) "pdf")
          (start-process "default-pdf-app" nil "open" lawlist-filename)))))
```

   TODO Fixme minor mode?

   http://www.emacswiki.org/emacs/FixmeMode http://www.emacswiki.org/emacs/fixme-mode.el

   Or see: hl-todo, and further packages like it, listed in hl-todo Help file:

- fic-ext-mode
- fic-mode
- fixme-mode
- fixmee
- see http://emacswiki.org/FixmeMode for more alternatives

   If you like this you might also like orglink. Mac-OS extension: Open file in finder

   From: http://stackoverflow.com/questions/20510333/in-emacs-how-to-show-current-file-in-finder

```
(defun open-finder ()
  (interactive)
  ;; IZ Dec 25, 2013 (3:25 PM): Making this work in dired:
  (if (equal major-mode 'dired-mode)
      (open-finder-dired)
      (let ((path
             (if (equal major-mode 'dired-mode)
                 (file-truename (dired-file-name-at-point))
               (buffer-file-name)))
            dir file)
```

```
        (when path
          (setq dir (file-name-directory path))
          (setq file (file-name-nondirectory path))
          (open-finder-1 dir file)))))

(defun open-finder-1 (dir file)
  (message "open-finder-1 dir: %s\nfile: %s" dir file)
  (let ((script
         (if file
             (concat
              "tell application \"Finder\"\n"
              " set frontmost to true\n"
              " make new Finder window to (POSIX file \"" dir "\")\n"
              " select file \"" file "\"\n"
              "end tell\n")
           (concat
            "tell application \"Finder\"\n"
            " set frontmost to true\n"
            " make new Finder window to {path to desktop folder}\n"
            "end tell\n"))))
    (start-process "osascript-getinfo" nil "osascript" "-e" script)))

;; own mod
(defun open-folder-in-finder (&optional dir)
  (interactive "DSelect folder:")
  (setq dir (expand-file-name dir))
  (let ((script
         (concat
          "tell application \"Finder\"\n"
          " set frontmost to true\n"
          " make new Finder window to (POSIX file \"" dir "\")\n"
          "end tell\n")))
    (start-process "osascript-getinfo" nil "osascript" "-e" script)))

(global-set-key (kbd "H-o") 'open-folder-in-finder)
```

Customization of Specific Authoring Modes

Scratchbooks for coding

logging tryout code

```
(defvar scratchpad-main-directory "1_SCRIPTS")

(defvar scratchpad-languages
```

```
  '(("emacslisp" .
              (:extension "el" :template-func make-el-template))
    ("supercollider" .
                  (:extension "scd" :template-func make-sc-template))
    ("markdown" .
     (:extension "md" :template-func make-md-template))
    ("shell" .
     (:extension "sh" :template-func make-sh-template))
    ("git" .
     (:extension "sh" :template-func make-sh-template))
    ("org-mode" .
     (:extension "org" :template-func make-org-template))))

(defun iz-scratchpad-menu (&optional folderp)
  (interactive "P")
  (let* ((menu (grizzl-make-index (mapcar 'car scratchpad-languages)))
         (language (grizzl-completing-read "Select language: " menu))
         (language-plist (cdr (assoc language scratchpad-languages))))
    (if folderp
        (dirtree (scratchpad-make-folder-name language) t)
      (apply
       (plist-get language-plist :template-func)
       (list
        language
        (read-no-blanks-input "Title? (only alpha-numeric, - and _ chars): " "")
        (plist-get language-plist :extension))))))

(file-name-sans-extension "/test/abcd.efgh")

(defun make-el-template (folder title extension)
  (let* (
         (full-path (scratchpad-make-full-path folder title extension))
         (file-name (file-name-nondirectory full-path))
         (package-name (file-name-sans-extension file-name)))
    (find-file full-path)
    (insert
     (concat
      ";;; package --- Summary\n\n"
      ";;; Commentary:\n\n"
      ";;; Code:\n\n()\n\n"
      ";;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;\n"
      "(provide '" package-name
      ")\n;;; " file-name " ends here"
      ))
```

```
    (goto-char 0)
    (search-forward "\(\)")
    (backward-char 1)))

(defun scratchpad-make-full-path (folder title extension)
  (concat (scratchpad-make-folder-name folder)
          (scratchpad-make-file-name title extension)))

(defun scratchpad-make-file-name (file-name extension)
  (concat title
          (format-time-string "_%y%m%d_%H-%M" (current-time))
          "."
          extension))

(defun scratchpad-find-file (folder file-name)
  (find-file (concat (scratchpad-make-folder-name folder) file-name)))

(defun scratchpad-make-folder-name (folder)
  (concat iz-log-dir scratchpad-main-directory "/" folder "-scratchpad/"))

(defun make-sc-template (folder title &optional extension)
  (unless extension (setq extension "scd"))
  (find-file
   (scratchpad-make-full-path folder title extension))
  (insert
   (concat "/* " (format-time-string "%c %Z") " */\n\n"
           "(\nServer.default.boot;\n)\n//:\n(\n"
           "~mySound = { | amp = 0.1 | WhiteNoise.ar(amp) }.play;\n)"
           ))
  (unless (sclang-get-process) (sclang-start)))

(defun make-md-template (folder title &optional extension)
  (unless extension (setq extension "md"))
  (find-file
   (scratchpad-make-full-path folder title extension))
  (insert
   (concat "# " title (format-time-string "\n(%c %Z)\n\n"))))

(defun make-sh-template (folder title &optional extension)
  (unless extension (setq extension "sh"))
  (find-file
   (scratchpad-make-full-path folder title extension))
  (insert
   (concat "#!/bin/sh\n# " title (format-time-string "(%c %Z)\n\n"))))
```

```
(defun make-org-template (folder title &optional extension)
  (unless extension (setq extension "org"))
  (find-file
   (scratchpad-make-full-path folder title extension))
  (insert
   (concat "#+TITLE: " title (format-time-string "\n#+DATE: %c %Z\n\n"))))

(global-set-key (kbd "H-h H-s") 'iz-scratchpad-menu)

(add-hook 'after-save-hook
          #'(lambda ()
              (and (save-excursion
                     (save-restriction
                       (widen)
                       (goto-char (point-min))
                       (save-match-data
                         (looking-at "^#!"))))
                   (not (file-executable-p buffer-file-name))
                   (shell-command (concat "chmod u+x " buffer-file-name))
                   (message
                    (concat "Saved as script: " buffer-file-name)))))
```

SuperCollider

sclang Setup

```
;;; Directory of SuperCollider support, for quarks, plugins, help etc.
(defvar sc_userAppSupportDir
  (expand-file-name "~/Library/Application Support/SuperCollider"))

;; Make path of sclang executable available to emacs shell load path
(add-to-list
 'exec-path
 "/Applications/SuperCollider/SuperCollider.app/Contents/Resources/")

;; Global keyboard shortcut for starting sclang
(global-set-key (kbd "C-c M-s") 'sclang-start)
;; overrides alt-meta switch command
(global-set-key (kbd "C-c W") 'sclang-switch-to-workspace)

;; Disable switching to default SuperCollider Workspace when recompiling SClang
(setq sclang-show-workspace-on-startup nil)

(require 'sclang)
```

SuperCollider-specific minor modes
Needs debugging: One of these modes breaks sclang-start:

```
;; Note: Paredit-style bracket movement commands d, u, f, b, n, p work
;; in sclang-mode without loading Paredit.
;; (add-hook 'sclang-mode-hook 'paredit-mode)
(add-hook 'sclang-mode-hook 'rainbow-delimiters-mode)
(add-hook 'sclang-mode-hook 'hl-sexp-mode)
(add-hook 'sclang-mode-hook 'electric-pair-mode)
(add-hook 'sclang-mode-hook 'yas-minor-mode)
(add-hook 'sclang-mode-hook 'auto-complete-mode)
;; sclang-ac-mode is included in sclang-extensions-mode:
;; (add-hook 'sclang-mode-hook 'sclang-ac-mode)
;; sclang-ac mode constantly tries to run code.
;; that can lead to loops that hang, for example constantly creating a view.
;; (add-hook 'sclang-mode-hook 'sclang-extensions-mode)
```

sclang keyboard shortcuts

```
;; Global keyboard shortcut for starting sclang
(global-set-key (kbd "C-c M-s") 'sclang-start)
;; Show workspace
(global-set-key (kbd "C-c C-M-w") 'sclang-switch-to-workspace)
```

Emacs Lisp

```
(add-hook 'emacs-lisp-mode-hook 'hl-sexp-mode)
(add-hook 'emacs-lisp-mode-hook 'hs-minor-mode)
(global-set-key (kbd "H-l h") 'hs-hide-level)
(global-set-key (kbd "H-l s") 'hs-show-all)

(add-hook 'emacs-lisp-mode-hook 'rainbow-delimiters-mode)
(require 'paredit) ;; smart edit parentheses
(require 'cl)
(require 'litable) ;; show lisp eval results in the buffer, interactively
(add-hook 'emacs-lisp-mode-hook 'paredit-mode)
(add-hook 'emacs-lisp-mode-hook 'turn-on-whitespace-mode)
(add-hook 'emacs-lisp-mode-hook 'auto-complete-mode)
(add-hook 'emacs-lisp-mode-hook 'turn-on-eldoc-mode)
;; H-C-i:
(define-key emacs-lisp-mode-map (kbd "H-TAB") 'icicle-imenu-command)
```

html/css/js

web-beautify. HTML, CSS, and JavaScript/JSON formatting `https://github.com/yasuyk/web-beautify`

Shell command, install js-beautify library:web

```
npm -g install js-beautify
```

Emacs sexp, install emacs web-beautify package:

```
(package-install 'web-beautify)
```

org-mode

binding for org show subtree

```
(eval-after-load 'org
    '(define-key org-mode-map (kbd "C-c C-x s") 'org-show-subtree))
```

Using ido for org-goto

```
(setq org-goto-interface 'outline-path-completion
      org-goto-max-level 10)
```

Working with icicles/ido-menu/lacarte in org-mode and elsewhere

1. lacarte/icicle-menu shortcut: H-C-i,

   ```
   ;; Previously bound only to org-mode map.
   (global-set-key (kbd "H-TAB") 'icicle-imenu)
   (global-set-key (kbd "H-C-l") 'lacarte-execute-menu-command)
   ```

2. making icicle-imenu and icicle-occur work with org-mode Following
   needs review! Fri, Nov 28 2014, 10:44 EET

   ```
   (defun org-icicle-occur ()
     "In org-mode, show entire buffer contents before running icicle-occur.
    Otherwise icicle-occur will not place cursor at found location,
    if the location is hidden."
     (interactive)
     (show-all)
     (icicle-occur (point-min) (point-max))
     (recenter 3))

   (eval-after-load 'org
     '(define-key org-mode-map (kbd "C-c '") 'org-edit-special))
   (eval-after-load 'org
     '(define-key org-mode-map (kbd "H-i") 'org-icicle-occur))
   (defun org-icicle-imenu (separate-buffer)
     "In org-mode, show entire buffer contents before running icicle-imenu.
   Otherwise icicle-occur will not place cursor at found location,
   if the location is hidden.
   ```

```
If called with prefix argument (C-u), then:
- open the found section in an indirect buffer.
- go back to the position where the point was before the command, in the
  original buffer."
  (interactive "P")
  (icicle-mode 1)
  (show-all)
  (let ((mark (point)))
    (icicle-imenu (point-min) (point-max) t)
    (cond (separate-buffer
           (org-tree-to-indirect-buffer)
           (goto-char mark))
          (t (recenter 4))))
  (icicle-mode -1))

(eval-after-load 'org
  '(define-key org-mode-map (kbd "C-c C-=") 'org-icicle-imenu))
(eval-after-load 'org
  '(define-key org-mode-map (kbd "C-c i m") 'org-icicle-imenu))

;; install alternative for org-mode C-c = org-table-eval-formula
;; which is stubbornly overwritten by icy-mode.
(eval-after-load 'org
  '(define-key org-mode-map (kbd "C-c C-x =") 'org-table-eval-formula))

;; Both eval-after-load and org-mode hook do not work for switching off
;; prelude mode, whitespace.  So using shortcuts as workaround:

(defun turn-off-whitespace-mode ()
  (interactive)
  (whitespace-mode -1))

(defun turn-off-icicle-mode ()
  (interactive)
  (icicle-mode -1))

(defun turn-off-prelude-mode ()
  (interactive)
  (prelude-mode -1))

(global-set-key (kbd "H-x w") 'turn-off-whitespace-mode)
(global-set-key (kbd "H-x p") 'turn-off-prelude-mode)
(global-set-key (kbd "H-x i") 'turn-off-icicle-mode)
```

```
(add-hook 'org-mode-hook
          (lambda ()
            (local-set-key (kbd "C-c M-=") 'org-table-eval-formula)
            (local-set-key (kbd "C-c '") 'org-edit-special)))

;;; ???? Adapt org-mode to icicle menus when refiling (C-c C-w)
;;; Still problems. Cannot use standard org refiling with icicles activated!
(setq org-outline-path-complete-in-steps nil)
```

3. Definitely switch prelude off in org mode, as it totally screws-up key bindings

   Especially in the case of Meta-shift-up and Meta-shift-down for spreadsheets. Have not figured out yet how to override those keys specifically.

```
(add-hook 'org-mode-hook
          (lambda ()
            (prelude-mode -1)))
(add-hook 'org-mode-hook 'prelude-off)
```

4. Providing alternatives for refile and copy using icicles

```
(defun org-refile-icy (as-subtree &optional do-copy-p)
  "Alternative to org-refile using icicles.
Refile or copy current section, to a location in the file selected with icicles.
Without prefix argument: Place the copied/cut section it *after* the selected section.
With prefix argument: Make the copied/cut section *a subtree* of the selected section.

Note 1: If quit with C-g, this function will have removed the section that
is to be refiled.  To get it back, one has to undo, or paste.

Note 2: Reason for this function is that icicles seems to break org-modes headline
buffer display, so onehas to use icicles for all headline navigation if it is loaded."
  (interactive "P")
  (outline-back-to-heading)
  (if do-copy-p (org-copy-subtree) (org-cut-subtree))
  (show-all)
  (icicle-imenu (point-min) (point-max) t)
  (outline-next-heading)
  (unless (eq (current-column) 0) (insert "\n"))
  (org-paste-subtree)
  (if as-subtree (org-demote-subtree)))

(defun org-copy-icy (as-subtree)
  "Copy section to another location in file, selecting the location with icicles.
```

```
  See org-refile-icy."
    (interactive "P")
    (org-refile-icy as-subtree t))

  (eval-after-load 'org
    '(define-key org-mode-map (kbd "C-c i r") 'org-refile-icy))
  (eval-after-load 'org
    '(define-key org-mode-map (kbd "C-c i c") 'org-copy-icy))
```

Use visual line, whitespace and windmove in org-mode

```
(add-hook 'org-mode-hook 'visual-line-mode)
(add-hook 'org-mode-hook 'turn-off-whitespace-mode)
(add-hook 'org-shiftup-final-hook 'windmove-up)
(add-hook 'org-shiftleft-final-hook 'windmove-left)
(add-hook 'org-shiftdown-final-hook 'windmove-down)
(add-hook 'org-shiftright-final-hook 'windmove-right)
```

Customize Org-mode display, including todo colors
Adapted from:

```
(setq org-startup-indented t) ;; auto-indent text in subtrees
(setq org-hide-leading-stars t) ;; hide leading stars in subtree headings
(setq org-src-fontify-natively t) ;; colorize source-code blocks natively
(setq org-todo-keywords
      '((sequence
         "!!!(1)"  ; next action
         "!!(2)"  ; next action
         "!(3)"  ; next action
         "TODO(t)"  ; next action
         "STARTED(s)"
         "WAITING(w@/!)"
         "TOBLOG(b)"  ; next action
         "SOMEDAY(.)" "|"
         "DONE(x@/@)"
         "CANCELLED(c@)"
         "OBSOLETE(o@)")
        (sequence
         "TODELEGATE(-)"
         "DELEGATED(d)"
         "DELEGATE_DONE(l!)")))

(setq org-todo-keyword-faces
      '(("!!!" . (:foreground "red" :weight bold))
        ("!!" . (:foreground "tomato" :weight bold))
```

```
        ("!" . (:foreground "coral" :weight bold))
        ("TODO" . (:foreground "LightSalmon" :weight bold))
        ("TOBLOG" . (:foreground "MediumVioletRed" :weight bold))
        ("STARTED" . (:foreground "DeepPink" :weight bold))
        ("WAITING" . (:foreground "gold" :weight bold))
        ("DONE" . (:foreground "SeaGreen" :weight bold))
        ("CANCELLED" . (:foreground "wheat" :weight bold))
        ("OBSOLETE" . (:foreground "CadetBlue" :weight bold))
        ("TODELEGATE" . (:foreground "DeepSkyBlue" :weight bold))
        ("DELEGATED" . (:foreground "turquoise" :weight bold))
        ("DELEGATE_DONE" . (:foreground "LawnGreen" :weight bold))
        ("WAITING" . (:foreground "goldenrod" :weight bold))
        ("SOMEDAY" . (:foreground "gray" :weight bold))))
```

Mobile Org

```
;; the rest of the setup was done by customizing the variables
;; org-mobile-directory and org-mobile-inbox-for-pull, and is in custom.el

(global-set-key (kbd "H-h m p") 'org-mobile-push)
(global-set-key (kbd "H-h m l") 'org-mobile-pull)
```

Following was tested, works OK, but is disabled for the moment:
`http://kenmankoff.com/2012/08/17/emacs-org-mode-and-mobileorg-auto-sync/`

```
(defun install-monitor (file secs)
  (run-with-timer
   0 secs
   (lambda (f p)
     (unless (< p (second (time-since (elt (file-attributes f) 5))))
       (org-mobile-pull)))
   file secs))

(defvar monitor-timer
  (install-monitor (concat org-mobile-directory "/mobileorg.org") 30)
  "Check if file changed every 30 s.")
```

line->headline

```
(defun org-headline-line ()
  "convert current line into headline at same level as above."
  (interactive)
  (beginning-of-line)
  (org-meta-return)
  (delete-char 1))
```

```
(eval-after-load 'org
  '(progn
     (define-key org-mode-map (kbd "C-M-<return>") 'org-headline-line)))
```

Agenda

1. Global key for org-agenda: C-c a

```
(global-set-key "\C-ca" 'org-agenda)
```

2. Add, remove, save agenda file list

```
(defvar org-agenda-list-save-path
  "~/.emacs.d/savefile/org-agenda-list.el"
"Path to save the list of files belonging to the agenda.")

(defun org-agenda-save-file-list ()
  "Save list of desktops from file in org-agenda-list-save-path"
  (interactive)
  (save-excursion
    (let ((buf (find-file-noselect org-agenda-list-save-path)))
      (set-buffer buf)
      (erase-buffer)
      (print (list 'quote org-agenda-files) buf)
      (save-buffer)
      (kill-buffer)
      (message "org-agenda file list saved to: %s" org-agenda-list-save-path))))

(defun org-agenda-load-file-list ()
  "Load list of desktops from file in org-agenda-list-save-path"
  (interactive)
  (save-excursion
    (let ((buf (find-file-noselect org-agenda-list-save-path)))
      (set-buffer buf)
      (setq org-agenda-files (eval (read (buffer-string))))
      (kill-buffer)
      (message "org-agenda file list loaded from: %s" org-agenda-list-save-path))))

(defun org-agenda-add-this-file-to-agenda ()
  "Add the file from the current buffer to org-agenda-files list."
  (interactive)
  (let (path)
    ;; (org-agenda-file-to-front) ;; adds path relative to user home dir
    ;; (message "Added current buffer to agenda files.")
    (let ((path (buffer-file-name (current-buffer))))
```

```
      (cond (path
        (add-to-list 'org-agenda-files path)
        (org-agenda-save-file-list)
        (message "Added file '%s' to agenda file list"
                 (file-name-base path)))
            (t (message "Cannot add buffer to file list. Save buffer first."))))))

(defun org-agenda-remove-this-file-from-agenda (&optional select-from-list)
  "Remove a file from org-agenda-files list.
If called without prefix argument, remove the file of the current buffer.
If called with prefix argument, then select a file from org-agenda-files list."
  (interactive "P")
  (let (path)
   (if select-from-list
       (let  ((menu (grizzl-make-index org-agenda-files)))
         (setq path (grizzl-completing-read "Choose an agenda file: " menu)))
     (setq path (buffer-file-name (current-buffer))))
   (setq org-agenda-files
         (remove (buffer-file-name (current-buffer)) org-agenda-files)))
  (org-agenda-save-file-list)
  (message "Removed file '%s' from agenda file list"
           (file-name-base (buffer-file-name (current-buffer)))))

(defun org-agenda-open-file ()
  "Open a file from the current agenda file list."
  (interactive)
  (let* ((menu (grizzl-make-index org-agenda-files))
         (answer (grizzl-completing-read "Choose an agenda file: " menu)))
    (find-file answer)))

(defun org-agenda-list-files ()
  "List the paths that are currently in org-agenda-files"
  (interactive)
  (let  ((menu (grizzl-make-index org-agenda-files)))
    (grizzl-completing-read "These are currently the files in list org-agenda-files. " menu)))

(defun org-agenda-list-menu ()
 "Present menu with commands for loading, saving, adding and removing
files to org-agenda-files."
 (interactive)
 (let* ((menu (grizzl-make-index
               '("org-agenda-save-file-list"
                 "org-agenda-load-file-list"
                 "org-agenda-list-files"
```

```
                  "org-agenda-open-file"
                  "org-agenda-add-this-file-to-agenda"
                  "org-agenda-remove-this-file-from-agenda")))
         (command (grizzl-completing-read "Choose a command: " menu)))
    (call-interactively (intern command))))

(global-set-key (kbd "H-a H-a") 'org-agenda-list-menu)
```

3. Calendar framework: Show org agenda in iCal-style layout

```
(require 'calfw-org)
```

4. Global key for cfw org calendar framework): C-c M-a

```
(global-set-key "\C-c\M-a" 'cfw:open-org-calendar)
(global-set-key "\C-c\C-xm" 'org-mark-ring-goto)
```

5. Insert DATE property

```
(defun org-set-date (&optional active property)
  "Set DATE property with current time.  Active timestamp."
  (interactive "P")
  (org-set-property
   (if property property "DATE")
   (cond ((equal active nil)
          (format-time-string (cdr org-time-stamp-formats) (current-time)))
         ((equal active '(4))
          (concat "["
                  (substring
                   (format-time-string (cdr org-time-stamp-formats) (current-time))
                   1 -1)
                  "]"))
         ((equal active '(16))
          (concat
           "["
           (substring
            (format-time-string (cdr org-time-stamp-formats) (org-read-date t t))
            1 -1)
           "]"))
         ((equal active '(64))
          (format-time-string (cdr org-time-stamp-formats) (org-read-date t t))))))

;; Note: This keybinding is in analogy to the standard keybinding:
;; C-c . -> org-time-stamp
```

```
  (eval-after-load 'org
    '(progn
       (define-key org-mode-map (kbd "C-c C-.") 'org-set-date)
       ;; Prelude defines C-c d as duplicate line
       ;; But we disable prelude in org-mode because of other, more serious conflicts,
       ;; So we keep this alternative key binding:
       (define-key org-mode-map (kbd "C-c d") 'org-set-date)))
```

6. Set DUE property with selected time/date

```
  (defun org-set-due-property ()
    (interactive)
    (org-set-property
     "DUE"
     (format-time-string (cdr org-time-stamp-formats) (org-read-date t t))))

  (eval-after-load 'org
    '(define-key org-mode-map (kbd "C-c M-.") 'org-set-due-property))
```

Class and Project notes, tags, diary

```
(setq org-clock-persist 'history)
(org-clock-persistence-insinuate)

(setq org-tag-alist
      '(
        ("home" . ?h)
        ("finance" . ?f)
        ("eastn" . ?e)
        ("avarts" . ?a)
        ("erasmus" . ?E)
        ("researchfunding" . ?r)))

(defvar iz-log-dir
  (expand-file-name
   "~/Dropbox/000WORKFILES/")
  "This directory contains all notes on current projects and classes")

(setq diary-file (concat iz-log-dir "PRIVATE/diary"))

(defadvice org-agenda (before update-agenda-file-list ())
  "Re-createlist of agenda files from contents of relevant directories."
  (iz-update-agenda-file-list)
  (icicle-mode 1))
```

```
(defadvice org-agenda (after turn-icicles-off ())
  "Turn off icicle mode since it interferes with some other keyboard shortcuts."
  (icicle-mode -1))

(ad-activate 'org-agenda)

(defadvice org-refile (before turn-icicles-on-for-refile ())
  "Turn on icicles before running org-refile.
Note: This piece of advice needs checking! Maybe not valid."
  (icicle-mode 1))

(defadvice org-refile (after turn-icicles-off-for-refile ())
  "Turn off icicle mode since it interferes with some other keyboard shortcuts."
  (icicle-mode -1))

(ad-activate 'org-refile)

(defun iz-diary-entry ()
  "Go to or create diary entry for date entered interactively."
  (interactive)
  (find-file (concat iz-log-dir "0_PRIVATE/DIARY.org"))
  (org-datetree-find-date-create
   (calendar-gregorian-from-absolute
    (org-time-string-to-absolute (org-read-date))))
  (org-show-entry))

(defun iz-update-agenda-file-list ()
  "Set value of org-agenda-files from contents of relevant directories."
  (setq org-agenda-files
        (let ((folders (file-expand-wildcards (concat iz-log-dir "*")))
              (files (file-expand-wildcards (concat iz-log-dir "*.org"))))
          (dolist (folder folders)
            (setq files
                  (append
                   files ;; ignore files whose name starts with dash (-)
                   (file-expand-wildcards (concat folder "/[!-]*.org")))))
          (-reject
           (lambda (f)
             (string-match-p "/\\." f))
           files)))
  (message "the value of org-agenda-files was updated"))

(defvar iz-last-selected-file
  nil
```

```
  "Path of file last selected with iz-org-file menu.
Used to refile to date-tree of last selected file.")

(defun iz-goto-last-selected-file ()
  (interactive)
  (if iz-last-selected-file
      (find-file iz-last-selected-file)
    (iz-find-file)))

(defun iz-refile-to-date-tree (&optional use-last-selected)
  "Refile using DATE timestamp to move to file-datetree.
If USE-LAST-SELECTED is not nil, refile to last selected refile target."
  (interactive "P")
  (let ((origin-buffer (current-buffer))
        (origin-filename (buffer-file-name (current-buffer)))
        (date (calendar-gregorian-from-absolute
               (org-time-string-to-absolute
                (or (org-entry-get (point) "CLOSED")
                 (org-entry-get (point) "DATE"))))))
    (org-cut-subtree)
    (if (and iz-last-selected-file use-last-selected)
        (find-file iz-last-selected-file)
      (iz-find-file))
    (org-datetree-find-date-create date)
    (move-end-of-line nil)
    (open-line 1)
    (next-line)
    (org-paste-subtree 4)
    (save-buffer)
    (find-file origin-filename)))

(defun org-process-entry-from-mobile-org ()
  "Get time from mobile-entry and put it in DATE property."
  (interactive)
  (org-back-to-heading 1)
  (next-line 1)
  (let ((time (cadr (org-element-timestamp-parser))))
    (org-entry-put nil "DATE" (plist-get time :raw-value)))
  (outline-next-heading))

(defun iz-get-and-refile-mobile-entries ()
  "Refile mobile entries to log buffer.
Use timestamp from mobile to refile under date-tree.
```

```
After finishing the refile operation, save a copy of the
processed file with a timestamp, and erase the contents of
from-mobile.org, to wait for next pull operation."
  (interactive)
 (org-mobile-pull)
 (let* ((mobile-file (file-truename "~/org/from-mobile.org"))
        (mobile-buffer (find-file mobile-file))
        (log-buffer (find-file (concat iz-log-dir "0_PRIVATE/DIARY.org"))))
   (with-current-buffer
       mobile-buffer
     (org-map-entries
      (lambda ()
        (let* ((timestamp
                (cdr (assoc "TIMESTAMP_IA" (org-entry-properties))))
               (date
                (calendar-gregorian-from-absolute
                 (org-time-string-to-absolute timestamp))))
          (org-copy-subtree)
          (with-current-buffer
              log-buffer
            (org-datetree-find-date-create date)
            (move-end-of-line nil)
            (open-line 1)
            (next-line)
            (org-paste-subtree 4)
            (org-set-property "DATE" (concat "<" timestamp ">"))
            (org-set-tags-to ":mobileorg:"))))))
   (copy-file
    mobile-file
    (concat
     (file-name-sans-extension mobile-file)
     (format-time-string "%Y-%m-%d-%H-%M-%S")
     ".org"))
   (with-current-buffer
       mobile-buffer
     (erase-buffer)
     (save-buffer))))

(defun iz-refile-notes-to-log ()
  "Refile notes entered from terminal with quick-entry to log file.
Get date from DATE property of entry and use it to refile the entry
in the log file under date-tree."
  (interactive)
 (let* ((notes-file (concat iz-log-dir "0_INBOX/notes.org"))
```

```
        (notes-buffer (find-file notes-file))
        (log-buffer (find-file (concat iz-log-dir "0_PRIVATE/DIARY.org"))))
   (with-current-buffer
       notes-buffer
     (org-map-entries
      (lambda ()
        (let* ((timestamp (org-entry-get (point) "DATE"))
               (date
               (calendar-gregorian-from-absolute
                (org-time-string-to-absolute timestamp))))
          (org-copy-subtree)
          (with-current-buffer
              log-buffer
            (org-datetree-find-date-create date)
            (move-end-of-line nil)
            (open-line 1)
            (next-line)
            (org-paste-subtree 4)
            (org-set-property "DATE" (concat "<" timestamp ">")))))))
   (copy-file
    notes-file
    (concat
     (file-name-sans-extension notes-file)
     (format-time-string "%Y-%m-%d-%H-%M-%S")
     ".org"))
   (with-current-buffer
       notes-buffer
     (erase-buffer)
     (save-buffer))))

(defun iz-insert-file-as-snippet ()
  (interactive)
  (insert-file-contents (iz-select-file-from-folders)))

(defun iz-select-file-from-folders ()
  (iz-org-file-menu (iz-select-folder)))

(defun iz-select-folder ()
  (let*
      ((folders (-select 'file-directory-p
                         (file-expand-wildcards
                          (concat iz-log-dir "*"))))
       (folder-menu (grizzl-make-index
                     (mapcar 'file-name-nondirectory folders)))
```

```
       (folder (grizzl-completing-read "Select folder:" folder-menu)))
    folder))

(defun iz-org-file-menu (subdir)
  (let*
      ((files
        (file-expand-wildcards (concat iz-log-dir subdir "/[!.]*.org")))
       (projects (mapcar 'file-name-sans-extension
                         (mapcar 'file-name-nondirectory files)))
       (dirs
        (mapcar (lambda (dir)
                  (cons (file-name-sans-extension
                                 (file-name-nondirectory dir)) dir))
                files))
       (project-menu (grizzl-make-index projects))
       (selection (cdr (assoc (grizzl-completing-read "Select file: " project-menu)
                              dirs))))
    (setq iz-last-selected-file selection)
    selection))

(defun iz-get-refile-targets ()
  (interactive)
  (setq org-refile-targets '((iz-select-file-from-folders . (:maxlevel . 2)))))

(defun iz-find-file (&optional dired)
  "open a file by selecting from subfolders."
  (interactive "P")
  (cond ((equal dired '(4))
         (dired (concat iz-log-dir (iz-select-folder))))
        ((equal dired '(16)) (dired iz-log-dir))
        ((equal dired '(64))
         (dirtree (concat iz-log-dir (iz-select-folder)) nil))
        ((equal dired '(256))
         (dirtree iz-log-dir nil))
        (t
         (find-file (iz-select-file-from-folders))
         (goto-char 0)
         (if (search-forward "*# -*- mode:org" 100 t)
             (org-decrypt-entries)))))

;; Following needed to avoid error message ls does not use dired.
(setq ls-lisp-use-insert-directory-program nil)
(require 'ls-lisp)
```

```
(defun iz-open-project-folder (&optional open-in-finder)
  "Open a folder associated with a project .org file.
Select the file using iz-select-file-from-folders, and then open folder instead.
If the folder does not exist, create it."
  (interactive "P")
  (let ((path (file-name-sans-extension (iz-select-file-from-folders))))
    (unless  (file-exists-p path) (make-directory path))
    (if open-in-finder (open-folder-in-finder path) (dired path))))

(defvar iz-capture-keycodes "abcdefghijklmnoprstuvwxyzABDEFGHIJKLMNOPQRSTUVWXYZ1234567890.,(){}!@#$%^&*

;; From: http://stackoverflow.com/questions/2321904/elisp-how-to-save-data-in-a-file

(defun dump-vars-to-file (varlist filename)
  "simplistic dumping of variables in VARLIST to a file FILENAME"
  (save-excursion
    (let ((buf (find-file-noselect filename)))
      (set-buffer buf)
      (erase-buffer)
      (dump varlist buf)
      (save-buffer)
      (kill-buffer))))

(defun dump (varlist buffer)
  "insert into buffer the setq statement to recreate the variables in VARLIST"
  (loop for var in varlist do
        (print (list 'setq var (list 'quote (symbol-value var)))
               buffer)))

(defvar iz-capture-template-history nil "something")

(defvar iz-capture-template-history-file
  (concat iz-log-dir "capture-template-history.el")
  "Store list of 10 last capture templates used.")

(defun iz-log (&optional goto)
  "Capture log entry in date-tree of selected file.
Select from menu comprized of 2 parts:
1. File selected from subfolders of log dir.
2. 20 latest files where a capture was performed.
"
  (interactive "P")
  (unless iz-capture-template-history
    (if (file-exists-p iz-capture-template-history-file)
```

```
        (load-file iz-capture-template-history-file)))
  (let*
      ((menu (grizzl-make-index
              (append
               (mapcar 'file-name-nondirectory
                       (-select 'file-directory-p
                                (file-expand-wildcards
                                 (concat iz-log-dir "*"))))
               (reverse (mapcar 'car iz-capture-template-history)))))
       (selection (grizzl-completing-read "Select log target:" menu)))
    (cond ((equal ":" (substring selection 0 1))
           (let ((org-capture-entry
                  (cdr (assoc selection iz-capture-template-history))))
             (org-capture goto)))
          (t
           (message "Selection: %s" selection)
           (message "Capture templates made from selection: %s"
                    (iz-make-log-capture-templates selection))
           (iz-make-log-capture-templates selection)
           (org-capture goto)))))

(defun org-capture-store-template-selection (&optional capt-template)
  "Keep list of 20 latest log files used."
  ;; (message "the arg is: %s" capt-template)
  (unless iz-capture-template-history
    (if (file-exists-p iz-capture-template-history-file)
        (load-file iz-capture-template-history-file)))
  (let ((key (concat ":" (car capt-template) "-" (cadr capt-template))))
    (setq iz-capture-template-history
          (-take 20
          (cons (cons key capt-template)
                (-reject (lambda (x) (equal key (car x)))
                         iz-capture-template-history)))))
  (dump-vars-to-file
   '(iz-capture-template-history)
   iz-capture-template-history-file)
  capt-template)

(advice-add
 'org-capture-select-template
 :filter-return
 'org-capture-store-template-selection)

;; old version:
```

```
(defun iz-log-old (&optional goto)
  "Capture log entry in date-tree of selected file."
  (interactive "P")
  (iz-make-log-capture-templates (iz-select-folder))
  (org-capture goto))

(defun iz-select-folder ()
  (let*
      ((folders (-select 'file-directory-p
                         (file-expand-wildcards
                          (concat iz-log-dir "*"))))
       (folder-menu (grizzl-make-index
                     (mapcar 'file-name-nondirectory folders)))
       (folder (grizzl-completing-read "Select folder:" folder-menu)))
    (file-name-nondirectory folder)))

(defun iz-make-log-capture-templates (subdir)
  "Make capture templates for selected subdirectory under datetree."
 (setq org-capture-templates
       (setq org-capture-templates
             (let* (
                    (files
                     (file-expand-wildcards
                      (concat iz-log-dir subdir "/[!-]*.org")))
                    (projects (mapcar 'file-name-nondirectory files))
                    (dirs
                     (mapcar (lambda (dir) (cons (file-name-sans-extension
                                                  (file-name-nondirectory dir))
                                                 dir))
                             files)))
               (-map-indexed (lambda (index item)
                               (list
                                (substring iz-capture-keycodes index (+ 1 index))
                                (car item)
                                'entry
                                (list 'file+datetree+prompt (cdr item))
                                "* %?\n :PROPERTIES:\n :DATE:\t%^T\n :END:\n\n%i\n"))
                             dirs)))))

(defun iz-todo (&optional goto)
  "Capture TODO entry in date-tree of selected file."
  (interactive "P")
  (iz-make-todo-capture-templates (iz-select-folder))
  (org-capture goto))
```

```
(defun iz-make-todo-capture-templates (subdir)
  "Make capture templates for project files"
 (setq org-capture-templates
       (setq org-capture-templates
             (let* (
                    (files
                     (file-expand-wildcards
                      (concat iz-log-dir subdir "/[a-zA-Z0-9]*.org")))
                    (projects (mapcar 'file-name-nondirectory files))
                    (dirs
                     (mapcar (lambda (dir) (cons (file-name-sans-extension
                                                  (file-name-nondirectory dir))
                                                 dir))
                             files)))
               (-map-indexed
                (lambda (index item)
                  (list
                   (substring iz-capture-keycodes index (+ 1 index))
                   (car item)
                   'entry
                   (list 'file+headline (cdr item) "TODOs")
                   "* TODO %?\n :PROPERTIES:\n :DATE:\t%U\n :END:\n\n%i\n"))
                dirs)))))

(defun iz-goto (&optional level)
  (interactive "P")
  (if level
      (setq org-refile-targets (list (cons (iz-select-file-from-folders) (cons :level level))))
    (setq org-refile-targets (list (cons (iz-select-file-from-folders) '(:maxlevel . 3)))))
  (org-refile '(4)))

(defun iz-refile (&optional goto)
  "Refile to selected file."
  (interactive "P")
  (setq org-refile-targets (list (cons (iz-select-file-from-folders) '(:maxlevel . 3))))
  (org-refile goto))

(defun iz-org-file-command-menu ()
  "Menu of commands operating on iz org files."
(interactive)
  (let* ((menu (grizzl-make-index
                '(
                  "iz-log"
```

```
                    "iz-todo"
                    "iz-refile-to-date-tree"
                    "iz-refile"
                    "iz-open-project-folder"
                    "iz-find-file"
                    "iz-goto"
                    "iz-goto-last-selected-file"
                    "org-agenda"
                    "iz-get-and-refile-mobile-entries"
                    "iz-refile-notes-to-log"
                    "iz-insert-file-as-snippet"
                    "iz-scratchpad-menu"
                    "iz-diary-entry"
                    "org-export-subtree-as-latex-with-header-from-file"
                    "org-export-subtree-as-pdf-with-header-from-file"
                    "org-export-buffer-as-latex-with-header-from-file"
                    "org-export-buffer-as-pdf-with-header-from-file"
                    )))
          (selection (grizzl-completing-read "Select command: " menu)))
    (eval (list (intern selection)))))

(global-set-key (kbd "H-h H-m") 'iz-org-file-command-menu)
(global-set-key (kbd "H-h H-h") 'iz-org-file-command-menu)
(global-set-key (kbd "H-h H-f") 'iz-find-file)
(global-set-key (kbd "H-h H-d") 'iz-open-project-folder)
(global-set-key (kbd "H-h H-l") 'iz-log)
(global-set-key (kbd "H-h L") 'iz-goto-last-selected-file)
(global-set-key (kbd "H-h H-i") 'iz-insert-file-as-snippet)
(global-set-key (kbd "H-h H-t") 'iz-todo)
(global-set-key (kbd "H-h H-r") 'iz-refile)
(global-set-key (kbd "H-h r") 'iz-refile-to-date-tree)
(global-set-key (kbd "H-h H-g") 'iz-goto)
(global-set-key (kbd "H-h H-c H-w") 'iz-refile)
(global-set-key (kbd "H-h H-c H-a") 'org-agenda)

;; Experimental:
(defun iz-make-finance-capture-template ()
  (setq org-capture-templates
        (list
         (list
          "f" "FINANCE"
          'entry
          (list 'file+datetree (concat iz-log-dir "projects/FINANCE.org"))
          "* %^{title}\n :PROPERTIES:\n :DATE:\t%T\n :END:\n%^{TransactionType}p%^{category}p%^{amount}p
```

```
        ))))
```

Org-Babel

1. Org-Babel: enable some languages

   Enable some cool languages in org-babel mode.

```
(org-babel-do-load-languages
 'org-babel-load-languages
 '((emacs-lisp . t)
   (sh . t)
   (ruby . t)
   (python . t)
   (perl . t)
   ))
```

2. Org-Babel: load current file

```
(defun org-babel-load-current-file ()
  (interactive)
  (org-babel-load-file (buffer-file-name (current-buffer))))

;; Note: Overriding default key binding to provide consistent pattern:
;; C-c C-v f -> tangle, C-c C-v C-f -> load
(eval-after-load 'org
  '(define-key org-mode-map (kbd "C-c C-v C-f") 'org-babel-load-current-file))
```

Orgmode latex customization

Note Mon, Dec 15 2014, 16:29 EET: XeLaTeX covers most needs that I have for western european languages, Greek and Japanese.

```
;;; Load latex package
(require 'ox-latex)

;;; Use xelatex instead of pdflatex, for support of multilingual fonts (Greek etc.)
;; Note: Use package polyglossia to customize dates and other details.
(setq org-latex-pdf-process
      (list "xelatex -interaction nonstopmode -output-directory %o %f"
            "xelatex -interaction nonstopmode -output-directory %o %f"
            "xelatex -interaction nonstopmode -output-directory %o %f"))

;; This is kept as reference. XeLaTeX covers all european/greek/asian needs.
;; It is the original setting for working with pdflatex:
;; (setq org-latex-pdf-process
;;  ("pdflatex -interaction nonstopmode -output-directory %o %f"
```

```
;;   "pdflatex -interaction nonstopmode -output-directory %o %f"
;;   "pdflatex -interaction nonstopmode -output-directory %o %f"))

;;; Add beamer to available latex classes, for slide-presentaton format
(add-to-list 'org-latex-classes
             '("beamer"
               "\\documentclass\[presentation\]\{beamer\}"
               ("\\section\{%s\}" . "\\section*\{%s\}")
               ("\\subsection\{%s\}" . "\\subsection*\{%s\}")
               ("\\subsubsection\{%s\}" . "\\subsubsection*\{%s\}")))

;;; Add memoir class (experimental)
(add-to-list 'org-latex-classes
             '("memoir"
               "\\documentclass[12pt,a4paper,article]{memoir}"
               ("\\section{%s}" . "\\section*{%s}")
               ("\\subsection{%s}" . "\\subsection*{%s}")
               ("\\subsubsection{%s}" . "\\subsubsection*{%s}")
               ("\\paragraph{%s}" . "\\paragraph*{%s}")
               ("\\subparagraph{%s}" . "\\subparagraph*{%s}")))

;; Reconfigure memoir to make a book (or report) from a org subtree
(add-to-list 'org-latex-classes
             '("section-to-book"
               "\\documentclass{memoir}"
               ("\\chapter{%s}" . "\\chapter*{%s}") ;; actually: BOOK TITLE!
               ("\\section{%s}" . "\\section*{%s}") ;; actually: Chapter!
               ("\\subsection{%s}" . "\\subsection*{%s}")
               ("\\subsubsection{%s}" . "\\subsubsection*{%s}")
               ("\\paragraph{%s}" . "\\paragraph*{%s}")))

;; Letter
(add-to-list 'org-latex-classes
             '("letter"
               "\\documentclass{letter}"
               ;; Should not use subsections at all!:
               ("\\chapter{%s}" . "\\chapter*{%s}")
               ("\\section{%s}" . "\\section*{%s}")
               ("\\subsection{%s}" . "\\subsection*{%s}")
               ("\\subsubsection{%s}" . "\\subsubsection*{%s}")
               ("\\paragraph{%s}" . "\\paragraph*{%s}")))

(add-to-list 'org-latex-classes
             '("newlfm-letter"
```

```
"\\documentclass[11pt,letter,dateno,sigleft]{newlfm}"
;; Should not use subsections at all!:
("\\chapter{%s}" . "\\chapter*{%s}")
("\\section{%s}" . "\\section*{%s}")
("\\subsection{%s}" . "\\subsection*{%s}")
("\\subsubsection{%s}" . "\\subsubsection*{%s}")
("\\paragraph{%s}" . "\\paragraph*{%s}")))
```

1. export subtree as latex with header selected from file template

   Todo:

   (a) **TODO** Define command for switching between xelatex/pdflatex

   (b) The code

```
(require 'org-attach)

(defvar latex-templates-path
  (file-truename "~/Dropbox/000WORKFILES/1_SNIPPETS_AND_TEMPLATES"))

(defvar latex-section-template
  '(("\\section\{%s\}" . "\\section*\{%s\}")
    ("\\subsection\{%s\}" . "\\subsection*\{%s\}")
    ("\\subsubsection\{%s\}" . "\\subsubsection*\{%s\}")))

(defvar org-latex-last-chosen-file-name nil
  "Path of last chosen latex template.")

(defun org-export-subtree-as-latex-with-header-from-file ()
  (interactive)
  (org-latex-export-with-file-template t t))

(defun org-export-subtree-as-pdf-with-header-from-file ()
  (interactive)
  (org-latex-export-with-file-template nil t))

(defun org-export-buffer-as-latex-with-header-from-file ()
  (interactive)
  (org-latex-export-with-file-template t nil))

(defun org-export-buffer-as-pdf-with-header-from-file ()
  (interactive)
  (org-latex-export-with-file-template nil nil))

(defun org-latex-export-with-file-template (&optional as-latex-buffer-p subtree-p)
  (let* (;; backup to restore original latex-classes after this operation:
```

```
    (org-latex-classes-backup org-latex-classes)
    (chosen-template-path (org-query-latex-template-path subtree-p))
    (this-buffers-latex-class
     (plist-get (org-export-get-environment 'latex subtree-p nil) :latex-class))
    latex-header
    (latex-sections
     (or ;; TODO: Add (org-latex-get-local-section-settings subtree-p) here.
      (cddr (assoc this-buffers-latex-class org-latex-classes))
      latex-section-templates)))
(when chosen-template-path
  (setq org-latex-last-chosen-file-name chosen-template-path)
  (setq latex-header
        (with-temp-buffer
          (insert-file-contents chosen-template-path)
          (concat
           "[NO-DEFAULT-PACKAGES]\n"
           "[NO-EXTRA]\n"
           "\n"
           (buffer-string))))
  ;; Create custom org-latex-classes to use this template:
  (setq org-latex-classes
        (list
         (append
          (list this-buffers-latex-class latex-header)
          latex-sections)))

  (if as-latex-buffer-p
      (org-latex-export-as-latex nil subtree-p nil nil)
    (let ((pdf-path (org-export-output-file-name ".pdf" subtree-p))
          (tex-path (org-export-output-file-name ".tex" subtree-p))
          (attach-path (org-file-or-subtree-attachment-dir subtree-p t)))
      (org-latex-export-to-pdf nil subtree-p nil nil)
      (copy-file pdf-path
                 (concat
                  attach-path
                  (file-name-nondirectory pdf-path))
                 t)
      (copy-file tex-path
                 (concat
                  attach-path
                  (file-name-nondirectory tex-path))
                 t)
      (copy-file chosen-template-path
                 (concat
```

```
                    attach-path
                    (file-name-nondirectory chosen-template-path))
                   t)
          (shell-command (concat
                          "open -a /Applications/Preview.app "
                          "\""
                          attach-path
                          (file-name-nondirectory pdf-path)
                          "\""))))
      ;; restore original latex classes:
      (setq org-latex-classes org-latex-classes-backup))))

;; TODO: (defun org-latex-get-local-section-settings (subtree-p) ...)

(defun org-query-latex-template-path (&optional subtree-p)
  "Get and set latex template path from menu of paths found in default folder.
Include:
1. list of template files found in latex-templates-path,
2. last used template org-latex-last-chosen-file-name,
3. template last chosen for export of this file or subtree,
4. and local copy of template used for export of this file or subtree.

Also:
1. Copy chosen template to attachment directory.
2. Store paths of chosen template and its copy as property.
"
  (let*
      ((paths (file-expand-wildcards (concat latex-templates-path "/*.tex")))
       (names-and-paths
        (mapcar
         (lambda (x)
           (cons (file-name-sans-extension (file-name-nondirectory x)) x))
         paths))
       (local-template-path (org-get-option-or-property "LATEX_TEMPLATE" subtree-p))
       (local-template-copy-path
        (org-get-option-or-property "LATEX_TEMPLATE_COPY" subtree-p))
       menu chosen-filename new-template-copy-path)
    (if org-latex-last-chosen-file-name
        (setq names-and-paths
              (append
               names-and-paths
               (list (cons (concat
                            "[Last chosen template:] "
                            (file-name-sans-extension
```

```
                          (file-name-nondirectory org-latex-last-chosen-file-name)))
                        org-latex-last-chosen-file-name)))))
    (if local-template-path
        (setq names-and-paths
              (append
               names-and-paths
               (list (cons (concat
                            "[Local template:] "
                            (file-name-sans-extension
                             (file-name-nondirectory local-template-path)))
                        org-latex-last-chosen-file-name)))))
    (if local-template-copy-path
        (setq names-and-paths
              (append
               names-and-paths
               (list (cons (concat
                            "[Copy of local template:] "
                            (file-name-sans-extension
                             (file-name-nondirectory local-template-copy-path)))
                        org-latex-last-chosen-file-name)))))
    (setq menu (grizzl-make-index (mapcar 'car names-and-paths)))
    (setq chosen-filename
          (cdr (assoc (grizzl-completing-read "Choose latex template: " menu)
                      names-and-paths)))
    (when chosen-filename
      (setq org-latex-last-chosen-file-name chosen-filename)
      (org-put-option-or-property "LATEX_TEMPLATE" chosen-filename subtree-p)
      (setq new-template-copy-path
            (concat
             (org-file-or-subtree-attachment-dir subtree-p t)
             (file-name-nondirectory chosen-filename)))
      (org-put-option-or-property "LATEX_TEMPLATE_COPY"
                                  new-template-copy-path subtree-p)
      (copy-file chosen-filename new-template-copy-path t))
    chosen-filename))

(defun org-file-or-subtree-attachment-dir (&optional subtree-p make-if-needed)
  "Return directory for attachments of whole file or subtree.
MAKE-IF-NEEDED indicates to create the directory if not present.

Whole file attachment directory is attachments/<file-name-sans-extension>/"
  (if subtree-p
      (concat (org-attach-dir t) "/")
    (let ((path
```

```
            (concat
             (file-name-directory (buffer-file-name))
             "attachments/"
             (file-name-sans-extension (file-name-nondirectory (buffer-file-name)))
             "/")))
        (unless (file-exists-p path)
          (if make-if-needed (make-directory path t)))
        path)))

(defun org-get-option-or-property (option &optional subtree-p)
  "Get value of the first definition of option OPTION in current buffer."
  (if subtree-p
      (org-entry-get (point) option)
      (org-with-wide-buffer
       (goto-char (point-min))
       (if (re-search-forward (org-make-options-regexp (list option)) nil t)
           (plist-get (cadr (org-element-at-point)) :value)))))

(defun org-put-option-or-property (option value &optional subtree-p)
  "Set option value in buffer."
  (if subtree-p
      (org-set-property option value)
    (org-with-wide-buffer
     (goto-char (point-min))
     (if (re-search-forward (org-make-options-regexp (list option)) nil t)
         (kill-whole-line)
       (goto-char (point-min)))
     (insert (concat "#+" option ": " value "\n")))))

(global-set-key (kbd "H-l H-p") 'org-export-subtree-as-pdf-with-header-from-file)
(global-set-key (kbd "H-l H-l") 'org-export-subtree-as-latex-with-header-from-file)
(global-set-key (kbd "H-l p") 'org-export-buffer-as-pdf-with-header-from-file)
(global-set-key (kbd "H-l l") 'org-export-buffer-as-latex-with-header-from-file)
```

Org-crypt: Encrypt selected org-mode entries

```
(require 'org-crypt)
(org-crypt-use-before-save-magic)
(setq org-tags-exclude-from-inheritance (quote ("crypt")))
;; GPG key to use for encryption
;; Either the Key ID or set to nil to use symmetric encryption.
(setq org-crypt-key nil)
```

org-reveal, ox-impress: Export slides for Reveal.js and impress.js from orgmode

Load org-reveal to make slides with reveal.js
https://github.com/yjwen/org-reveal/ https://github.com/kinjo/org-impress-js.el

```
(require 'ox-reveal)
(require 'ox-impress-js)
```

Folding and unfolding, selecting headings

1. Extra shortcut: Widen

```
(eval-after-load 'org
  '(define-key org-mode-map (kbd "H-W") 'widen))
```

2. Macro: toggle drawer visibility for this node

See: http://stackoverflow.com/questions/5500035/set-custom-keybinding-for-specific-emacs-mode

```
(fset 'org-toggle-drawer
   (lambda (&optional arg) "Keyboard macro." (interactive "p") (kmacro-exec-ring-item (quote ([671088

(eval-after-load 'org
  '(define-key org-mode-map (kbd "C-c M-d") 'org-toggle-drawer))
```

3. Toggle folding of current item (Command and keyboard command)

```
(defun org-cycle-current-entry ()
  "toggle visibility of current entry from within the entry."
  (interactive)
  (save-excursion)
  (outline-back-to-heading)
  (org-cycle))

(eval-after-load 'org
  '(define-key org-mode-map (kbd "C-c C-/") 'org-cycle-current-entry))
```

4. Keyboard Command Shortcut: Select heading of this node (for editing)

Note: outline-previous-heading (C-c p) places the point at the beginning of the heading line. To edit the heading, one has to go past the * that mark the heading. org-select heading places the mark at the beginning of the heading text and selects the heading, so one can start editing the heading right away.

```
(defun org-select-heading ()
  "Go to heading of current node, select heading."
  (interactive)
```

```
    (outline-previous-heading)
    (search-forward (plist-get (cadr (org-element-at-point)) :raw-value))
    (set-mark (point))

    (beginning-of-line)
    (search-forward " "))

  (eval-after-load 'org
    '(define-key org-mode-map (kbd "C-c C-h") 'org-select-heading))
```

Encryption

```
(require 'org-crypt)
(org-crypt-use-before-save-magic)
(setq org-tags-exclude-from-inheritance (quote ("crypt")))
;; GPG key to use for encryption
;; Either the Key ID or set to nil to use symmetric encryption.
(setq org-crypt-key nil)
```

Create menu for org-mode entries (lacarte lets you reach it from the keyboard, too)

```
(add-hook 'org-mode-hook
          (lambda () (imenu-add-to-menubar "Imenu")))
(setq org-imenu-depth 3)
```

Property shortcuts for collaboration: From-To

Note: searchable both with org-mode match: C-c / p and with icicles search, org-icicle-occur or icicle-occur, here: C-c C-'

```
(defun org-from ()
  "Set property 'FROM'."
  (interactive)
  (org-set-property "FROM" (ido-completing-read "From whom? " '("ab" "iz"))))

(defun org-to ()
  "Set property 'TO'."
  (interactive)
  (org-set-property "TO" (ido-completing-read "To whom? " '("ab" "iz"))))

(eval-after-load 'org
  '(define-key org-mode-map (kbd "C-c x f") 'org-from))
(eval-after-load 'org
  '(define-key org-mode-map (kbd "C-c x t") 'org-to))
```

OBSOLETE fname-find-file-standardized: Consistent multi-component filenames

```
(defvar fname-parts-1-2 nil)
(defvar fname-part-3 nil)
(defvar fname-root "~/Dropbox/000Workfiles/2014/")
(defvar fname-filename-components
  (concat fname-root  "00000fname-filename-components.org"))

(defun fname-find-file-standardized (&optional do-not-update-timestamp)
  (interactive "P")
  (unless fname-part-3 (fname-load-file-components))
  (setq *grizzl-read-max-results* 40)
  (let* ((root fname-root)
         (index-1 (grizzl-make-index
                   (mapcar 'car fname-parts-1-2)))
         (name-1 (grizzl-completing-read "Part 1: " index-1))
         (index-2 (grizzl-make-index (cdr (assoc name-1 fname-parts-1-2))))
         (name-2 (grizzl-completing-read "Part 2: " index-2))
         (index-3 (grizzl-make-index fname-part-3))
         (name-3 (grizzl-completing-read "Part 3: " index-3))
         (path (concat root name-1 "_" name-2 "_" name-3 "_"))
         (candidates (file-expand-wildcards (concat path "*")))
         extension-index extension final-choice)
    (setq final-choice
          (completing-read "Choose file or enter last component: " candidates))
    (cond ((string-match (concat "^" path) final-choice)
           (setq path final-choice))
      (t
       (setq extension (ido-completing-read
                        "Enter extension:" '("org" "el" "html" "scd" "sc" "ck")))
       (setq path (concat path final-choice
                          (format-time-string "_%Y-%m-%d-%H-%M" (current-time))
                          "." extension))))
    (find-file path)
    (unless do-not-update-timestamp
     (set-visited-file-name
      (replace-regexp-in-string
       "_[0-9]\\{4\\}-[0-9]\\{2\\}-[0-9]\\{2\\}-[0-9]\\{2\\}-[0-9]\\{2\\}"
       (format-time-string "_%Y-%m-%d-%H-%M" (current-time)) path)))
    (kill-new (buffer-file-name (current-buffer)))))

(defun fname-load-file-components (&optional keep-buffer)
  (interactive "P")
  (let ((buffer (find-file fname-filename-components)))
    (fname-load-file-components-from-buffer buffer)
    (unless keep-buffer (kill-buffer buffer)))
```

```
  (message "file component list updated"))

(defun fname-load-file-components-from-buffer (buffer)
  (set-buffer buffer)
  (setq fname-parts-1-2 nil)
  (setq fname-part-3 nil)
  (org-map-entries
   (lambda ()
     (let ((plist (cadr (org-element-at-point))))
       (cond
        ((equal (plist-get plist :level) 2)
         (setq fname-parts-1-2
               (append fname-parts-1-2
                       (list (list (plist-get plist :raw-value))))))
        ((equal (plist-get plist :level) 3)
         (setcdr (car (last fname-parts-1-2))
                 (append (cdar (last fname-parts-1-2))
                         (list (plist-get plist :raw-value))))))))
   "LEVELS1_2")
  (org-map-entries
   (lambda ()
     (let ((plist (cadr (org-element-at-point))))
       (when
           (equal 2 (plist-get plist :level))
         (setq fname-part-3
               (append fname-part-3 (list (plist-get plist :raw-value)))))))
   "LEVEL3"))

(defun fname-edit-file-components ()
  (interactive)
  (find-file fname-filename-components)
  (add-to-list 'write-contents-functions
               (lambda ()
                 (fname-load-file-components-from-buffer (current-buffer))
                 (message "Updated file name components from: %s" (current-buffer))
                 (set-buffer-modified-p nil)))
  ;; Debugging:
  (message "write-contents-functions of file %s are: %s"
           (buffer-file-name) write-contents-functions))
  (defun fname-menu ()
  (interactive)
  (let ((action (ido-completing-read
                 "Choose action: "
                 '("fname-edit-file-components"
```

```
                "fname-load-file-components"
                "fname-find-file-standardized"))))
    (funcall (intern action))))

(global-set-key (kbd "H-f f") 'fname-find-file-standardized)
(global-set-key (kbd "H-f m") 'fname-menu)
(global-set-key (kbd "H-f e") 'fname-edit-file-components)
(global-set-key (kbd "H-f l") 'fname-load-file-components)
```

Macro: toggle drawer visibility for this section;
See: http://stackoverflow.com/questions/5500035/set-custom-keybinding-for-specific-emacs-mode

```
(fset 'org-toggle-drawer
   (lambda (&optional arg) "Keyboard macro." (interactive "p") (kmacro-exec-ring-item (quote ([67108896
```

```
(eval-after-load 'org
  '(define-key org-mode-map (kbd "C-c M-d") 'org-toggle-drawer))
(eval-after-load 'org
  '(define-key org-mode-map (kbd "C-c C-'") 'org-edit-special))
```

org-export for docpad

```
(defun org-html-export-as-html-body-only ()
  "Export only the body. Useful for using the built-in exporter of Org mode
with the docpad website framework."
    (interactive)
    (let ((path
           (concat
            (file-name-sans-extension (buffer-file-name))
            ".html")))
      (message path)
      (org-html-export-as-html
          nil ;; async
          nil ;; subtreep
          nil ;; visible-only
          t   ;; body only
          ;; ext-plist (not given here)
          )
      (write-file path)
      (message (format "written to path: %s" path))))

(global-set-key (kbd "H-e H-b") 'org-html-export-as-html-body-only)
```

Internal: Load org-pm

```
(org-babel-load-file "/Users/iani/Documents/Dev/Emacs/org-publish-meta/org-pm.org")
```

Magit (git for emacs) Add git repositories)

Magit config: Manage git repos from inside emacs

```
(setq magit-repo-dirs
      '(
        "~/Dropbox/000WORKFILES/org"
        "~/Documents/Dev"
        "~/.emacs.d/personal"
))
```

Load Default Desktop

Note: This must be after the TODO config, otherwise any Org mode buffers loaded here will not inherit the TODOD config.

```
(bmkp-desktop-jump "startup")
```

Shell Scripts

Saving my config to github, bitbucket, gitlab

```
cd ~/Documents/Dev/Emacs/my-emacs-prelude-config/
git add --all .
git commit -am "Saved on: `date`"
# echo "================================================================="
echo "!!!!!!!!!!!!!!!! Created new git commit !!!!!!!!!!!!!!!!"
echo "========================================== Pushing to github"
git push origin master
echo "======================== Saved repo to github.  Now pushing to bitbucket."
git push bitbucket master
echo "======================== Saved repo to bitbucket.  Now pushing to gitlab."
git push gitlab master
echo "======================== Saved repo to gitlab. Backup completed."
```

Notes

DONE *Try Orgstruct, Outshine, Outorg or Poporg for elisp files instead of Babel*

`http://orgmode.org/worg/org-tutorials/org-outside-org.html`

Capture for project logs in one folder

About folder "project-logs"

This folder, project-logs, is for monitoring the progress of individual projects. Each project has a separate file.

Methodology

Keep using the global file log.org for fast logging of all entries, regarding any project. To update the status of a project in the individual project files of the present folder, go to the global log.org file and copy sections to the related individual project files. For example, if file log.org contains a section related to project "proj1", then copy that section to file "proj1.org". Therefore keep both the original entries in log and the copied entries in the individual project files in the present folder ("project-logs"). In file log.org it may be useful to mark those entries that have been copied with a property such as "COPIED" set to true.

Keyboard shortcuts, custom functions

Copy subtree to other subtree.

New functions to program:

C-c Like org-capture, but modified in the following points:

1. Use a grizzle-style vertical menu for selecting the target of refile (instead of a single-key shortcut)
2. Create the menu of refile targets by scanning the project-logs folder (instead of statically from a list hard-coded in emacs-lisp).
3. Possibly allow several variants of capture:
   (a) date-tree-prompt style (file+datetree+prompt)
   (b) "entry" type, target (file "path/to/file")
   (c) "entry" type, target id or file+headline

Implementation

Implementation is underway in file scratch/disperse.org

Postamble

```
;;; org-pm.el ends here
```